REMARKS

Claims 1, 3-4, 6, 9, 11-16, and 18-20 are amended. Claims 2, 7, and 10 are canceled without prejudice or disclaimer. No new matter is added by these amendments. Claims 1, 3-6, 8-9, and 11-21 are pending. Applicant respectfully requests reconsideration and allowance of all claims in view of the amendments above and the remarks that follow.

Claim Objections

Claim 7 is objected to because "[t]he letter 's' is missing on the word 'exceed.'" Applicant respectfully traverses this grounds for rejection because "any remaining class-type checks" (the subject) is plural and "exceed" (the verb) is also plural, so the phrase has proper subject-verb agreement since the subject and verb are both plural.

Claim 8 is objected to because "[a] colon is missing after 'means for calculating' and there should be semicolons where there are commas." Applicant respectfully traverses these grounds for rejection because adding a semicolon after "means for calculating" would cause the clause to read "means for calculating: the number based on ..." which would be a grammatically incorrect use of a semicolon.

Applicant further respectfully traverses these grounds for rejection because a cost of the inline code, a cost of an out-of-line class-type check, and a number of times the inline code fails" are not separate elements; instead, they are all factors on which the means for calculating is based.

Claim 13 is objected to because "there should be semicolons where there are commas." Applicant respectfully traverse these grounds for rejection because "a cost of the inline code, a cost of the out-of-line function call, and a number of times the inline code fails" are not separate elements; instead, they are all factors on which the calculating is based.

Claim 16 is objected to because "there should an 's' on the word 'comprise'." Applicant respectfully traverses these grounds for rejection because "instructions" (the subject) and "comprise" (the verb) are both plural, so the phrase has proper subject-verb agreement since the subject and verb are both plural. "On the processor" is a prepositional phrase, which is not the subject and which is ignored for the purposes of subject-verb agreement.

Claim 16 is further objected to because "there should be semicolons where there are commas." Applicant respectfully traverses these grounds for rejection because the electronic device comprises two elements: a processor and a storage device. Thus, a semicolon follows "processor." In contrast, the "calculating," "generating," "sorting," and "generating" are instructions, and are not elements at the level of the processor and the storage device, so the "calculating," "generating," "sorting," and "generating" are separated by commas and not semicolons.

Claim 18 is objected to because "there should be an 's' on the word 'comprise.'" Applicant respectfully traverses these grounds for rejection because "instructions" (the subject) and "comprise" (the verb) are both plural, so the phrase has proper subject-verb agreement.

Claims 18-20 are objected to because 'there should be a colon after 'comprises.'" Claims 18-20 are amended to add a semicolon.

Claim Objections under 35 U.S.C. 112

Claims 14, 20, and 21 are objected to under 35 U.S.C. 112 for insufficient antecedent basis for "object type." Claims 14 and 20 are amended to provide antecedent basis.

Claim Rejections under 35 U.S.C. 101

Claims 11-15 are rejected under 35 U.S.C. 101 for claiming a signal bearing medium. Claims 11-15 are amended to recite a storage medium, which is statutory under 35 U.S.C. 101.

Rejections under 35 U.S.C. 102 and 103

Claims 11 and 13 are rejected under 35 U.S.C. 102(b) as being anticipated by Shaylor (US 6,760,907). Claims 1-9 and 12 are rejected under 35 U.S.C. 103(a) as unpatentable over Shaylor in view of Tsuboi (US 6,848,098). Claims 10 and 16-19 are rejected under 35 U.S.C. 103(a) as unpatentable over Shaylor in view of Tsuboi and Lueh (U.S. 6,658,657). Claims 14, 20, and 21 are rejected under 35 U.S.C. 103(a) as unpatentable over Shaylor. Applicant respectfully submits that the claims are patentable over the references for the reasons argued below.

Claim 1 recites: "calculating a number of class-type checks at a site in a method that minimizes a cost of inlining; generating inline code for the number of the class-type checks for the site in the method; sorting the inline code based on a frequency of the class types; and generating an out-of-line function call for any remaining class-type checks at the site that exceed the number and that are not handled by the inline code."

Shaylor does not teach or suggest "calculating a number of class-type checks at a site in a method that minimizes a cost of inlining," as recited in claim 1 because Shaylor at column 6, lines 23-25, merely describes "four categories of method calls in which the invention is particularly useful." Shaylor does not describe calculating a number of its categories at a particular site in a particular method. Thus, Shaylor does not teach or suggest "calculating a number of class-type checks at a site in a method that minimizes a cost of inlining," as recited in claim 1.

Tsuboi at column 5, lines 32-33, merely describes "the number of executions of a callable program." Tsuboi does not teach or suggest that its callable program has a class type, that a class type is ever checked, or that a number of class-type checks is every calculated. Thus, the Tsuboi "number" is unrelated to a number of class-type checks at a site in a method, and is in fact a completely different type of number: a number of executions. Thus, the hypothetical combination of Shaylor and Tsuboi teaches away from "calculating a number of class-type checks at a site in a method that minimizes a cost of inlining," as recited in claim 1.

Lueh, at column 5, lines 53-54 describes "memory 560 maintains the number of times a call site is frequently called." Thus, Lueh's "number" is also unrelated to a number of class-type checks, and Lueh has no notion of a class-type check. Hence, Lueh does not teach or suggest calculating a number of class-type checks, and the hypothetical combination of Shaylor, Tsuboi, and Lueh teaches away from "calculating a number of class-type checks at a site in a method that minimizes a cost of inlining," as recited in claim 1.

Shaylor does not teach or suggest "generating inline code for the number of the class-type checks for the site in the method," as recited in claim 1 because Shaylor does not have a "number of class-type checks" for which to generate inline code. Instead, Shaylor decides whether or not "method calls can always be called directly or inlined" (Shaylor at column 6, lines 50-53) based on the Shaylor four categories (Shaylor at column 6, lines 23-25).

In Shaylor's first case (column 6, lines 25-27), "the receiver object's class ... is unambiguously known," and the choice between a direct call and inlining is "a matter of the appropriate code generation to meet the needs of a particular application." Shaylor at column 6, lines 60-62. Thus, in Shaylor's first case, Shaylor does not describe any specific technique for deciding whether to inline or direct call, so Shaylor does not teach or suggest "generating inline code for the number of the class-type checks for the site in the method," where the number was calculated by the "calculating" as recited in claim 1.

In Shaylor's "second case" (column 6, lines 32-33), "the receiver class is known to be variable," so Shaylor "would generate code that implements a traditional function call." Shaylor at column 6, lines 63-65. Alternatively, Shaylor identifies "all possible classes" as "cases in a 'switch statement' so that the target method's code from each possible class is inlined." Shaylor at column 7, lines 1-3. Thus, in Shaylor's second case, Shaylor either generates a function call or inlines code from each possible class. Hence, once again, Shaylor's decision-making process for generating inline code is unrelated to a number of class-type checks, so Shaylor does not teach or suggest "generating inline code for the number of the class-type checks for the site in the method," where the number was calculated by the "calculating" as recited in claim 1.

In Shaylor's "third case" (column 6, lines 41-43), "the receiver class is identified as one that more often than not is of a particular type," so Shaylor tests whether the class is correct, and "if the class is not the correct one for the method, a virtual function call can be made. Alternatively, case three methods can be inlined." (Shaylor at column 7, lines 8-11.) Hence, Shaylor's decision-making process for generating inline code is unrelated to a number of class-type checks calculated to minimize cost, so Shaylor does not teach or suggest "generating inline code for the number of the class-type checks for the site in the method," where the number of class-type checks to inline was calculated to minimize the cost of inlining, as recited in claim 1.

In Shaylor's "fourth case" (column 6, lines 46-47), "the class has never been subclassed in any previous execution," so Shaylor calls the method directly, but if the class is subsequently subclassed, then Shaylor tests the receiver and makes the virtual function call if it is not correct. If it is correct, Shaylor branches back to the main method code. (Shaylor at column 7, lines 15-22.) Hence, Shaylor's decision-making process for generating inline code is unrelated to a number of class-type checks, so Shaylor does not teach or suggest "generating inline code for the number of the class-type checks for the site in the method," where the number was calculated by the "calculating" as recited in claim 1.

Tsuboi, at column 5, lines 32-35 describes that its "judging unit 6 determines whether or not the number of executions of a callable program part exceeds a predetermined threshold, and notifies the optimization unit 7 of the excess of the number of executions."

Thus, Tsuboi does not teach or suggest "generating inline code for the number of the class-type checks for the site in the method," where the number was calculated by the "calculating" as recited in claim 1 because the Tsuboi "number" is a "number of executions" and not a "number of the class-type checks," as recited in claim 1. Further, Tsuboi does not generate inline code for a calculated number of the class-type checks (as recited in claim 1) in response to determining the Tsuboi number of executions. Instead, Tsuboi merely "notifies the optimization unit 7 of the excess of the number of executions," which moves loop invariant operations, removes unnecessary instructions, and performs in-line expansion of functions, as described at Tsuboi at column 8, lines 40-63. Tsuboi does not teach or suggest that its "in-line expansion of functions" are class-type checks or that the number of its functions that are expanded inline was calculated to minimize a cost of inlining, as recited in claim 1. Thus, Tsuboi teaches away from "generating inline code for the number of the class-type checks," as recited in claim 1.

Lueh, at column 5, lines 53-54 describes that the "memory 560 maintains the number of times a call site is frequently called." Thus, Lueh's "number" is unrelated to a number of class-type checks, and Lueh has no notion of a class-type check. Hence, Lueh does not teach or suggest generating inline code for the number of the class-type checks, and the hypothetical combination of Shaylor, Tsuboi, and Lueh teaches away from "generating inline code for the number of the class-type checks for the site in the method," as recited in claim 1.

Shaylor does not teach or suggest "sorting the inline code based on a frequency of the class types," as recited in claim 1. Instead, Shaylor at column 6, lines 12-16 merely describes that the "As execution proceeds, profiler 210 stores profile data ... indicating whether a particular class has been subclassed, and indicating whether one class type of a

particular class is most likely to be the correct class type at any particular method invocation." The profile data is not shown in Shaylor (Shaylor at column 6, lines 5-6) and is not part of any code. Thus, Shaylor's profile data is not inline code, Shaylor does not sort any inline code based on its profile data, and "indicating whether a particular class has been subclassed" does not teach or suggest sorting inline code. Thus, Shaylor does not teach or suggest "sorting the inline code," nor does Shaylor teach or suggest "sorting the inline code based on a frequency of the class types," as recited in claim 1.

Tsuobi does not teach or suggest "sorting the inline code based on a frequency of the class types," as recited in claim 1. Instead, Tsuboi, at column 5, lines 32-35 merely describes that its "judging unit 6 determines whether or not the number of executions of a callable program part exceeds a predetermined threshold, and notifies the optimization unit 7 of the excess of the number of executions." The Tsuboi optimization unit 7 raises the optimization level of a program part when the optimization unit 7 receives the notification of the excess number of executions, as described by Tsuboi at column 5, lines 39-44. Tsuboi performs its optimization by moving loop invariant operations outside of loops, by removing load instructions when the register already contains the value, and by in-line expansion as described by Tsuboi at column 8, lines 45-64. Thus, Tsuboi never sorts its program parts based on a frequency of the class types because moving loop invariant operations, removing unnecessary load instructions, and in-line expansion are not sorting and because Tsuboi performs its optimization in response to a number of executions exceeding a threshold and not based on a frequency of class types. Thus, Tsuboi does not teach or suggest "sorting the inline code based on a frequency of the class types," as recited in claim 1.

Lueh also does not teach or suggest "sorting the inline code based on a frequency of the class types," and Lueh was not relied upon for such by the Office Action. Thus, the hypothetical combination of Shaylor, Tsuboi, and Lueh does not teach or suggest "sorting the inline code based on a frequency of the class types," as recited in claim 1.

With respect to claims 14, 20, and 21, the Office Action took Official Notice that an object is an instance of a class. Applicant respectfully objects to this Official Notice because claims 14, 20, and 21 do not recite that an object is an instance of a class. Instead, claim 14 recites: "calculating the number of class-type checks based on a count of object types encountered at the site at runtime," claim 20 recites: "calculating the number based on a count of object types encountered at the site at runtime," and claim 21 recites: "incrementing the count at runtime."

Independent claims 6, 11, and 16 include similar elements as argued above for claim 1 and are patentable over the references for similar reasons. Claims 3-5, 8-9, 12-15, and 17-21 are dependent on claims 1, 6, 11, and 16, respectively, and are patentable for the reasons argued above, plus the elements in the claims.

Conclusion

Applicant respectfully submits that the claims are in condition for allowance and notification to that effect is requested. The Examiner is invited to telephone Applicant's attorney (651-645-7135) to facilitate prosecution of this application.

If necessary, please charge any additional fees or credit overpayment to Deposit Account No. 09-0465.

Respectfully submitted,

John G. Nistler, et al.

By their Representative,

Owen J. Gamon

Date: January 31, 2007

Owen J. Gamon
Reg. No. 36,143
(651) 645-7135

IBM Corporation
Intellectual Property Law
Dept. 917, Bldg. 006-1
3605 Highway 52 North
Rochester, MN 55901

CERTIFICATE OF TRANSMISSION UNDER 37 C.F.R. 1.8

I hereby certify that the enclosed or attached correspondence is being transmitted via facsimile to the Commissioner for Patents, 571-273-8300, on January 31, 2007.

Owen J. Gamon

Owen J. Gamon